Real-Time Fair Resource Allocation in Distributed Software Defined Networks

Zaid Allybokus, Konstantin Avrachenkov, Jérémie Leguay, Lorenzo Maggi

Real-Time Fair Resource Allocation in Distributed Software Defined Networks

Zaid Allybokus*†, Konstantin Avrachenkov†, Jérémie Leguay*, Lorenzo Maggi*
*Huawei Technologies, France Research Center
{zaid.allybokus, jeremie.leguay, lorenzo.maggi}@huawei.com
†INRIA, Sophia Antipolis
avratch@inria.fr

***Abstract*—The performance of computer networks relies on how bandwidth is shared among different flows. Fair resource allocation is a challenging problem particularly when the flows evolve over time. To address this issue, bandwidth sharing techniques that quickly react to the traffic fluctuations are of interest, especially in large scale settings with hundreds of nodes and thousands of flows. In this context, we propose a distributed algorithm based on the Alternating Direction Method of Multipliers (ADMM) that tackles the fair resource allocation problem in a distributed SDN control architecture. Our ADMM-based algorithm continuously generates a sequence of resource allocation solutions converging to the fair allocation while always remaining feasible, a property that standard primal-dual decomposition methods often lack. Thanks to the distribution of all computer intensive operations, we demonstrate that we can handle large instances in real-time.**

***Index Terms*—Software-Defined Networking; Fair Resource Allocation; Alternating Direction Method of Multipliers; α-Fairness; Distributed Algorithms; Distributed SDN Control Plane.**

I. Introduction

Software Defined Networking (SDN) technologies are radically transforming network architectures by offloading the control plane (e.g., routing, resource allocation) to powerful remote platforms that gather and keep a local or global view of the network status in real-time and push consistent configuration updates to the network equipment. The computation power of SDN controllers fosters the development of a new generation of control plane architecture that uses compute-intensive operations. Initial design of SDN architectures [24] had envisioned the use of one central controller. However, for obvious scalability and resiliency reasons, the industry has quickly realized that the SDN control plane needs to be partially distributed in large network scenarios [11]. Hence, although *logically* centralized, in practice the control plane may consist of multiple controllers each in charge of a SDN domain of the network and operating together, in a *flat* [22] or *hierarchical* [7] architecture.

In this paper, we study the problem of computing a globally *fair* (in the sense of α-fairness defined by Mo and Walrand in [16], see Section III) resource allocation in a distributed SDN scenario, where the control plane is distributed over several domain controllers. In this context, flows transiting in the network typically correspond to traffic aggregates of a customer or a class of customers. We consider the traffic engineering use case where the size of flows evolves over time and the bandwidth reserved to each of them has to be quickly adjusted towards the novel fair solution.

In distributed SDN architectures, each controller has full information about its own domain. Moreover, it can communicate with adjacent peer controllers and/or with a central, upper-layer controller entity. However, exchanges between controllers are expensive in terms of communication delay and overhead [20]. This technological limitation translates directly into an algorithmic constraint: distributed algorithms for SDN have a limited budget in terms number of iterations to reach convergence.

A second crucial property for any distributed algorithm for SDN is responsiveness. In fact, the network state may be affected by abrupt changes, e.g., flow size variation, flow arrival/departure, link/node congestion. In this case, convergence for the previous network state may not even be attained when a change occurs in the system. For this reason, it is often preferable to have a quick access to a good quality solution rather than a provably asymptotically optimal solution with poor convergence rate. Hence, it is crucial that the resource allocation computed by a distributed algorithm is *feasible, thus implementable*, at any iteration.

To recap, we identify two main requirements for a distributed algorithm for fair resource allocation, namely *i)* converging to a "good" fair solution in a small number of iterations and *ii)* producing *feasible* solutions at all iterations.

We claim that none of the current methods that allocate resources in an SDN scenario is able to achieve the two aforementioned goals at the same time. Local mechanisms such as Auto-Bandwidth [18] have been proposed to greedily and distributedly adjust the allocated bandwidth to support time-varying IP traffic in *Multi Protocol Label Switching* (MPLS) networks. Auto-Bandwidth successfully tackles goal *ii)* but not *i)*, as it neither ensures fairness nor optimizes resources globally. Also, classic primal-dual algorithms have been proposed to solve the α-fair resource allocation problem in distributed SDN scenarios, as in [15]. However, primal-dual algorithms are known to fail at providing feasible solutions at any iteration step, thus they fail at achieving goal *ii)*.

Recently in the optimization research community, the *Alternating Direction Method of Multipliers* (ADMM) [3] has captured the attention for its separability and fast convergence properties. We claim that ADMM offers new and yet unexploited possibilities to tackle concurrently the goals *i), ii)*. Indeed, in this paper we show how ADMM serves our purposes, by allowing all controllers to handle their own domains simultaneously, while still converging to a global optimum in the fashion of a general distributed consensus

problem.

Main contributions: We develop an ADMM-based method (FD-ADMM, Algorithm 2) for the α-fair resource allocation problem over a distributed SDN control plane. It iteratively produces resource allocations that converge to the α-fair optimal allocation. Heavy computations, requiring projections on polytopes, can be massively parallelized on a link-by-link basis (Algorithm 2, line 7) in each domain. This yields a convergence rate that does not depend on the partitioning of the network into domains.

We show that our FD-ADMM algorithm can function in real-time, as *i)* close-to-optimal solutions are available since the *very first* iterations and *ii)* feasible allocations are available at *all* iterations (Proposition 1), a property that standard primal-dual decomposition methods generally lack. This permits to adjust within up to a few milliseconds the bandwidth of flows that evolve quickly and need immediate response.
Moreover, we show how to achieve a near-optimal convergence rate by providing an explicit and adaptive tuning for the FD-ADMM penalty parameter (Scheme 1).

Finally, we benchmark FD-ADMM with the State-of-the-Art (SoA) approach for computing α-fair resource allocations in distributed scenarios in [15], which is based on the Lagrangian dual splitting method. We show that our algorithm outperforms the SoA in terms of convergence rate, feasibility preservation and hence, overall, real-time responsiveness.

The remainder of this paper is organized as follows. Section II surveys the related work around the fair resource allocation problem. Section III formulates the α-fair resource allocation problem and presents the basic centralized application of ADMM. Section IV introduces FD-ADMM, our distributed ADMM-based algorithm that benefits from the distribution of SDN controllers over multiple domains. FD-ADMM is based on a reformulation of the problem in the fashion of a general distributed consensus problem. Section V discusses on a near-optimal tuning of the penalty parameter in FD-ADMM. Section VI provides simulations that validate our approach and finally, Section VII concludes the paper.

For the sake of conciseness, we deferred some of our proofs to the technical report in [1].

II. Related work

The concept of fair resource allocation has been a central topic in networking. Particularly, *max-min* fairness[1] has been the classic resource sharing principle [2] and has been studied extensively. The concept of *proportional fairness* and its weighted variants were introduced in [10]. Later, a spectrum of fairness metrics including the two former ones was introduced by Mo and Walrand in [16] as the family of α-fair utility functions.

Some early notable works on max-min fairness include [4], where the authors propose an asynchronous distributed algorithm that communicates explicitly with the sources and pays some overhead in exchange for more robustness and faster convergence. Later in [21], a distributed algorithm is defined for the weighted variant of max-min fair resource allocation problem in MPLS networks, based on the well-known property that an allocation is max-min fair if and only if each *Label-Switched Path* (LSP) either admits a *bottleneck link* amongst its used links or meets its maximal bandwidth requirement (see Definition 4 there of a bottleneck link). The problem of Network Utility Maximization (NUM) was also addressed with standard decomposition methods that could give efficient and very simple algorithms based on gradient ascent schemes performing their update rules in parallel. In this context, Voice [25], then McCormick et al. [15], tackle the α-fair resource allocation problem with a gradient descent applied to the dual of the problem.

In these works, no mention is made on the potential (in fact, systematic) feasibility violation of the sequences generated by those algorithms, which is a crucial matter in distributed SDN settings. Regarding this topic, the authors of [12] employ damping techniques to avoid transient infeasibility while reaching the max-min fair point, but cannot guarantee feasibility at all times, especially in dynamic settings. Also motivated by this, more recently the authors of [23] provide a feasibility preserving version of Kelly's methodology in [10]. Their algorithm introduces a slave that gives at each (master) iteration an optimal solution of a weighted proportionally fair resource allocation problem that is explicitly addressed in only the two cases of polymatroidal and flow aggregating networks. In fact, our paper contributes to this problem by proposing an efficient *real-time* version of the slave process, for any topology, preserving feasibility at each (slave) iteration. Amongst approximative approaches, one can quote the very recent work [14] where a multiplicative approximation for $\alpha \neq 1$ and additive approximation for $\alpha = 1$ is provably obtained in poly-logarithmic time in the problem parameters. Moreover, starting from any point, the algorithm reaches feasibility within poly-logarithmic time and remains feasible forever after. The algorithm described in our paper solves the problem optimally and reaches feasibility as from the first iteration from any starting point.

The work around ADMM is currently flourishing. The $O(\frac{1}{n})$ best known convergence rate of ADMM [8] failed to explain its empirical fast convergence until very recently, for instance in [6], where global linear convergence rates are established in four scenarios of the strongly convex case. ADMM is also well-known for its performance that highly depends on the parameter tuning, namely, the penalty parameter ρ (or reciprocal penalty parameter $\lambda = 1/\rho$) in the augmented Lagrangian formulation (see Section III-B below). An effective use of this class of algorithms cannot be decoupled from an accurate parameter tuning, as convergence can be extremely slow otherwise. Thus, in the same paper [6], the authors provide a linear convergence proof that yields a convergence rate in a closed form that can be optimized with respect to the problem parameters. Therefore, thanks to these works, we derive in our paper a near-optimal tuning of ADMM for the α-fair resource allocation problem. Several papers use the distributivity of ADMM to design efficient distributed algorithms solving consensus formulations for e.g. model predictive control [17] and resource allocation in wireless

[1] A resource allocation strategy is said to be max-min fair if no route can increase its allocation while remaining feasible without penalizing another route that has a smaller or equal allocation

virtual networks [13] but do not address this fundamental detail.

To the best of our knowledge, we are the first to show how ADMM can help designing real-time distributed algorithms for computing α-fair resource allocations in distributed settings. We are also the first to exhibit a near-optimal convergence rate of ADMM in this situation with our reciprocal penalty parameter adaptation scheme.

III. Fair resource allocation problem

In this section, we reformulate the α-fair resource allocation problem as a convex optimization problem. Then, we start off with our algorithm design by presenting C-ADMM, an algorithm that solves our problem in a centralized fashion and that will be helpful to design our distributed algorithm.

A. Problem reformulation

Let R be a set of connection requests over a network with a set J of capacitated links. Each link $j \in J$ has a total capacity of $C_j \in \mathbf{R}_+$. Each request r is represented by a route containing a subset of J that, without any confusion, we still denote as r. With some abuse of notation, we write $j \in r$ or $r \in j$ to say that link j belongs to the route r, or route r goes through link j, respectively. Given the set of requests and their corresponding utility function f_r, the network allocates bandwidth to all the requests in order to maximize the overall utility $f = \bigoplus_r f_r$, while satisfying feasibility, i.e., the link capacity constraints. Denote by x_r the capacity allocated to route r, and let $x = (x_r)_{r\in R}$. Then, we have the classic capacity constraint in matricial form:

$$Ax \leq C \tag{1}$$

where $A = (a_{jr})_{jr}$ is the link-route incidence binary matrix:

$$a_{jr} = \begin{cases} 1 & \text{if } j \in r \\ 0 & \text{otherwise.} \end{cases}$$

Our aim is to compute an α-fair capacity allocation x:

$$\max_{x\geq 0, Ax\leq C} f^\alpha(x) \tag{2}$$

where the α-fair utility function f^α is defined according to the Mo and Walrand's classic characterization in [16], that we report below.

Definition 1 ((w,α)-fairness, [16])**.** Let $F \subset \mathbf{R}_+^n$ be a non-empty feasible set not reduced to $\{0\}$. Let $w \in \mathbf{R}_+^n$ and $x^* \in F$. We say that x^* is (w,α)-*fair* (or simply α-fair when there is no confusion on w) if the following holds:

$$\forall r \in [1,n], \quad x_r^* > 0 \quad \text{and} \quad \forall x \in F, \quad \sum_{r=1}^n w_r \frac{x_r - x_r^*}{x_r^{*\alpha}} \leq 0.$$

Equivalently, x^* is (w,α)-fair if, and only if x^* maximizes the α-fair utility function f^α defined over $F - \{0\}$:

$$f^\alpha(x) = \sum_{r=1}^n f_r^\alpha(x_r),$$

where $f_r^\alpha(x_r) = \begin{cases} w_r \frac{x_r^{1-\alpha}}{1-\alpha}, & \alpha \neq 1, \\ w_r \log(x_r), & \alpha = 1. \end{cases}$

The success of α-fairness is due to its generality: in fact, for $\alpha = 0, 1, 2, \infty$ it is equivalent to max-throughput, proportional fairness, min-delay, and max-min fairness, respectively. We observe that the α-fair utility functions are non-decreasing, strictly concave, non-identically equal to $-\infty$, and upper semi-continuous. It is well-known that under these conditions, the function f^α admits a unique maximizer over any convex closed non-empty set.

From now on, we adopt the convex optimization terminology. Define for each $r \in R$ the convex cost function $g_r : x_r \mapsto g_r(x_r) := -f_r(x_r)$. Then, $g := \bigoplus_r g_r = -f^\alpha$ is a convex closed proper[2] function over $\mathbf{R}_+^{|R|}$. Let us introduce ι as the indicator function of the convex closed set $\{Ax \leq C, x \geq 0\}_x$:

$$\iota(x) = \begin{cases} 0 & \text{if } Ax \leq C \\ \infty, & \text{otherwise.} \end{cases}$$

Then our α-fair problem can equivalently be formulated as the following convex program:

$$\min_{x,z} \sum_{r\in R} g_r(x_r) + \iota(z), \tag{3}$$

$$\text{s.t.} \quad x - z = 0. \tag{4}$$

B. ADMM as an augmented Lagrangian splitting

Let us begin by recalling to the reader the basic principles of the *Alternating Direction Method of Multipliers* (ADMM), applied to our α-fair problem. To this aim, the augmented Lagrangian with penalty $\lambda^{-1} > 0$ for problem (3-4) writes[3]

$$L_{\lambda^{-1}}(x,z,u) = g(x) + \iota(z) + u^{\mathrm{T}}(x-z) + \frac{1}{2\lambda}||x-z||^2 \tag{5}$$

where u is the vector of Lagrange multipliers. The method of multipliers consists in the following update rules, where the superscript k denotes an iteration count:

$$(x^{k+1}, z^{k+1}) = \arg\min_{x,z} L_{\lambda^{-1}}(x,z,u^k) \tag{M1}$$

$$u^{k+1} = u^k + \frac{1}{\lambda}(x^{k+1} - z^{k+1}). \tag{M2}$$

The main idea in alternating directions is in fact to decouple the variables (x,z) in the optimization stage M1: instead of a global optimization over (x,z), we only optimize $L_{\lambda^{-1}}$ with respect to the variable x, then, given the new update of x, we optimize $L_{\lambda^{-1}}$ with respect to z. Before stating the corresponding update rules of ADMM, let us first remind the following Fact.

Fact 1 ([3])**.** *Let $h : \mathbf{R}^n \to \bar{\mathbf{R}} = \mathbf{R} \cup \{\infty\}$ be a closed proper convex function. The set $\mathrm{dom}(h)$ denotes the domain of h, that is the set upon which h takes real values. Assume $\mathrm{dom}(h) \neq \emptyset$. Then, the following facts hold:*

(i) For $u \in \mathbf{R}^n$, $\lambda \in \mathbf{R}_+^*$, the minimization problem

$$u_\lambda^* = \arg\min_x \left\{ h(x) + \frac{1}{2\lambda}||u-x||^2 \right\}$$

[2] *closed* stands for lower semi-continuous and *proper* means non-identically equal to ∞

[3] $a^{\mathrm{T}}b$ is the Euclidean product of a and b and $||\cdot||$ the Euclidean norm.

admits a unique solution. The (λ-scaled) proximal operator *of h is the well-defined map* $\text{prox}_{\lambda h} : u \to u^*_\lambda$.

(ii) Assume that h takes the form $h(x,y) = h_1(x) + h_2(y)$, for $(x,y) \in \mathbf{R}^p \times \mathbf{R}^{n-p}$ (write $h = h_1 \bigoplus h_2$) where h_1, h_2 are both closed, proper and convex. Then, for $(u,v) \in \mathbf{R}^p \times \mathbf{R}^{n-p}$, $\text{prox}_{\lambda h}(u,v) = \big(\text{prox}_{\lambda h_1}(u), \text{prox}_{\lambda h_2}(v)\big)$.

(iii) Assume that h is the indicator function of a closed convex non-empty set F. Then $P_F := \text{prox}_{\lambda h}$ is the Euclidean projection onto F.

The definition of a proximal operator being set, a straightforward calculus shows that we have:

$$\forall x, u \quad \arg\min_z L_{\lambda^{-1}}(x, z, u) = \text{prox}_{\lambda \iota}(x + \lambda u)$$

$$\forall z, u \quad \arg\min_x L_{\lambda^{-1}}(x, z, u) = \text{prox}_{\lambda g}(x - \lambda u).$$

ADMM can thus be expressed in the proximal (λ-scaled) form, which we refer to as *Centralized* ADMM (C-ADMM).

Algorithm 1 Centralized ADMM (C-ADMM)

Input: Initial values z, v
1: **while** a suitable termination condition is not met **do**
2: $\quad x \leftarrow \text{prox}_{\lambda g}(z - v)$
3: $\quad z \leftarrow P(x + v)$
4: $\quad v \leftarrow v + x - z$
5: **end while**

In Algorithm 1, $P = \text{prox}_{\lambda \iota}$ is the projection on $\{Ax \leq C, x \geq 0\}_x$, and $v = \lambda u$ the λ-scaled dual variable. Now, the first step of Algorithm 1 (line 2) can be separated thanks to the separability property of the objective function, see Fact 1. In fact, g is fully separable, as $g(x) = \sum g_r(x_r)$. Thus, the proximal update of line 2 takes the trivially parallelized form:

$$\forall r \quad x_r^{k+1} = \text{prox}_{\lambda g_r}(z_r^k - u_r^k) \tag{6}$$

such that each local variable x_r can be computed separately.

Through expression (6), we are thus able to provide an efficient update rule for x, provided that the separate proximal computations are inexpensive. However, two main issues arise.

Main issues with C-ADMM: *a)* First, an update of the variable z in line 3 of Algorithm 1 requires full knowledge of the projection mapping, which in turn requires full information on the capacity set of the network. Thus, this global update rule represents an important limiting factor to the design of a fully distributed algorithm, which is our main design interest here to follow the distribution of SDN control planes.

b) Moreover, although the convergence of C-ADMM may only require some tens of iterations (see Section VI for further details), it may be slow in terms of computation time due the successive application of a projection algorithm that would not scale with respect to the problem size. This also gives rise to a double loop algorithm where each iteration requires the convergence of an inner process that can be time-consuming. Indeed, computing the projection of a generic point onto a closed convex non-empty polyhedron is in general non-trivial. Hence, for general polyhedra, one has to operate alternate projections, summon quadratic programming solvers or use iterative algorithms such as the one in [9].

We address issues *a,b)* in the next section, where we propose FD-ADMM, a distributed version of C-ADMM.

IV. The general consensus form of ADMM: an efficient distributed algorithm design

In this section, we show how to alleviate the cost of the global projection sub-routine in C-ADMM (line 3) by decomposing the formulation with respect to the network links of each SDN domain in the fashion of a consensus problem, and present FD-ADMM. As stated at end of Section III, the global knowledge of the topology and the computational effort required by the projection step (line 3) of C-ADMM are not affordable in the distributed SDN control plane. Thus, the decomposition permits to respect the locality of the different domain controllers that now handle the projections link by link efficiently and in parallel. The decomposition into domains can be orchestrated by the SDN architect without any constraint. Unavoidably though, domains will need to exchange information as routes may traverse different domains.

A. Preliminaries

We organize the network into several domains $J_p, p = 1 \dots P$ such that $(J_p)_p$ forms a partition of the set of links J. Let R_p be the set of routes traversing the domain J_p via some link $j \in J_p$. More formally, $R_p = \{r \in R : \exists j \in J_p \text{ s.t. } j \in r\}$. Hence, $(R_p)_p$ forms a covering of R. Let ι_j denote the indicator function for link $j \in J_p$, i.e.,

$$\iota_j(x) = \begin{cases} 0 & \text{if } \sum_{r \in j} x_r \leq C_j \\ \infty & \text{otherwise.} \end{cases} \tag{7}$$

Also, let us define $S_j := \text{dom}(\iota_j)$. Thus, for each $j \in J$, S_j is the (convex, closed) capacity set of the link j. Finally, for $j \in J$ and $z \in \mathbf{R}^R$, PROJECTION(j, z) denotes the Euclidean projection of z onto S_j.

B. Consensus form

We can now reformulate our objective to a fully separable form. For ease of notation, the variable x will be written z_0 and we define $R_0 = R$. We also define an additional variable, $\tilde{z}_r$, that will represent the consensus value of z_{0r} found for each route r over all the domains handling the route r. We write $I_r = \{q \in [0, P] \quad r \in R_q\}$ to design the set of domain indices (including index 0) which r belongs to. In the same fashion as in Section III-B, we plug the feasibility constraints into the objective. Each constraint being now handled separately, we can formulate Problem (3), (4) as follows:

$$\min \sum_{r \in R} g_r(z_{0r}) + \sum_{j \in J} \iota_j(z_0). \tag{8}$$

In order to obtain a separable objective and fully benefit from the separability property in Fact 1, we artificially create a copy of the variable z_0 for each link j. This variable will be handled by the unique domain J_p containing j. For each j, let $z_j \in \mathbf{R}^{|R|}$ be the copy of z_0 for link j.

Creating a complete copy of all the variables for each domain is, nevertheless, of no use. Each domain indeed only

needs information and manipulation over the only variables associated with the routes that they handle completely (the route is included in the domain's links) or partially (the route meets other domains). Now, ι_j actually depends only on the sub-variable $(z_j)_{R_j} \stackrel{\text{def}}{=} (z_{jr})_{r \in R_j}$. We erase all the information that is irrelevant to region J_p: $z_j \in \mathbf{R}^{R_j}$. We can thus write the objective as follows:

$$\min G(z) = \sum_{r \in R} g_r(z_{0r}) + \sum_{j \in J} \iota_j((z_j)_{R_j}). \tag{9}$$

To sum up, *we have artificially separated the objective function by creating a minimal number of copies of the primal variable z_0 in order to fully distribute the problem*. Now, instead of a global resource allocation variable, several copies of the variable account for how its value is perceived by each link of each domain. To enforce an intra- (local) and inter- (global) domain consistent value of the appropriate allocation, consensus constraints are added to the problem. This new formulation can be interpreted as a multi-agent consensus problem formulation where route r has cost g_r, and link j has cost ι_j. As we separated the global objective on purpose, the separability property of the proximal operator thus gives the following:

$$\text{prox}_{\lambda G}(u) = \left((\text{prox}_{\lambda g_r}(u_{0r}))_r, (\textsc{Projection}(j, u_j))_j\right).$$

These considerations permit next to write our final distributed consensus model where each agent only has access to local information.

C. Fast Distributed ADMM

We can finally distribute ADMM by putting into practice the tricks described in the previous section. Then, the general consensus form of the problem can be expressed as follows.

$$\min \sum_{r \in R} g_r(z_{0r}) + \sum_{j \in J} \iota_j(z_j) \tag{10}$$

$$z_{jr} = z_{lr} \quad \forall r \in R_j \cap R_l \quad \forall j, l \in \{0\} \cup J \tag{11}$$

where $z_j = (z_{jr})_{r \in R_j} \in \mathbf{R}_+^{|R_j|}$. By applying ADMM to this formulation and using again Fact 1 we obtain, after some simplification, Algorithm 2 (Fast Distributed (FD)-ADMM).
To update the consensus variables $\tilde{z}_r$, we exploit the fact that the Euclidean projection of a point $y \in \mathbf{R}^n$ onto the diagonal is simply its average $\frac{1}{n}\sum y_i \mathbf{1}$. Hence, if I denotes the indicator function of the feasible set (11), we have:

$$\forall r \in R \quad \text{prox}_{\lambda I}(u)_r = \frac{1}{|J_r|+1}\left(\sum_{l \in r} u_{lr} + u_{0r}\right).$$

This yields the simple update rules at lines 4 and 10[4].
Notably, even in the distributed case, each domain p can compute at each iteration a *globally* feasible allocation z_{*r} for each of the routes $r \in R_p$ (see Proposition 1).
Communication among domain controllers: In FD-ADMM, *only domains that do share a route together have to communicate*. The communication procedures among the domain controllers are described at lines 2 and 10. In these steps, the domains gather from and broadcast to adjacent domains the sole information related to routes that they share in common. In particular, domains are blind to routes that do not traverse them, and can keep their internal routes secret from others. In details, after each iteration of the algorithm, each domain J_p receives the minimal information from other domains such that J_p is still able to compute a local value z_{pr} and a locally feasible value z_{*pr}. Next, J_p send them back to neighboring domains I_r that r traverses.

[4]These updates rules are also simplified using the straightforward fact that the sum $\sum_{l \in r} u_{lr}$ is constant. It can thus be fixed to 0 by initialization.

Algorithm 2 Fast Distributed ADMM (FD-ADMM)

1: **procedure OF** DOMAIN p
Input: Reciprocal penalty parameter λ, $(g_r)_{r \in R_p}$
2: RECEIVE $z_{qr}, z_{*qr} \quad \forall q \in I_r \quad \forall r \in R_p$
3: ENFORCE $z_{*r} = \min_{q \in I_r} z_{*qr} \quad \forall r \in R_p$
4: $\tilde{z}_r \leftarrow \frac{1}{|J_r|+1}\left(\sum_{q \in I_r} z_{qr} + z_{0r}\right) \quad \forall r \in R_p$
5: **for** $j \in J_p \cup \{0\}$ **do**
6: $u_{jr} \leftarrow u_{jr} + z_{jr} - \tilde{z}_r \quad \forall r \in R_j$
7: $z_j \leftarrow \textsc{Projection}(j, \tilde{z} - u_j)$
8: **end for**
9: $z_{0r} \leftarrow \text{prox}_{\lambda g_r}(\tilde{z}_r - u_{0r}) \quad \forall r \in R_p$
10: SEND $z_{pr} = \sum_{j \in J_r \cap J_p} z_{jr}$ and $z_{*pr} = \min_{j \in J_p} z_{jr}$ to domains $q \in I_r \quad \forall r \in R_p$
11: **end procedure**

Communication overhead: In terms of overhead, we can easily evaluate the number of floats transmitted between each domain at each iteration. At each communication, domain J_p must transmit z_{pr} and z_{*pr} for each $r \in R_p$ to each other domain that r traverses. The variable z_0 does not need to be centralized or transmitted between controllers. Each domain controller may actually have a copy z_0 and perform the (low-cost) computation of their update rule (see line 9 in Algorithm 2) locally. Hence, domain p transmits in total $2\sum_{q \neq p} |R_p \cap R_q|$ floats to the set of its peers. As a comparison, in a distributed implementation of the algorithm given in [15] and stated in Section VI, each domain p transmits in total $\sum_{q \neq p} |\{j \in J_p, \exists r \in R_q \text{ s.t. } j \in r\}|$ floats to the set of its peers, which is bounded by $(P-1)|J_p|$ as $|R|$ grows.
Feasibility preservation: A potential drawback of the distributed approach is the potential feasibility violation by the iterate $\tilde{z}^k$. However, we have the following positive result.

Proposition 1. *FD-ADMM provides a sequence of feasible points that converges to the optimum.*

Proof. Consider the iteration number k and drop the superscript k for lightness. For any link j, we have by line 7 of Algorithm 2 that z_j is feasible in link j. That is, $\sum_{r \in j} z_{jr} \leq C_j$. Define $z_{*r} = \min_{j \in J_r} z_{jr}$. Then, for each link j:

$$\sum_{r \in j} z_{*r} \leq \sum_{r \in j} z_{jr} \leq C_j. \tag{12}$$

Thus, no capacity is violated by the allocation z_{*r}. At the optimum, the consensus is reached. Thus $(z_{*r}^k)_k$ is a feasible sequence that converges to the optimum. □

The number z_{*r} introduced in Proposition 1 above in fact corresponds to the introduced variable of the same name FD-ADMM. Thus, in a certain way, for sufficiently loaded and communicating domains (i.e. the $|R_p \cap R_q|$ are large enough) we sacrifice some overhead (counted on a per iteration basis) compared to standard dual methods, but in exchange for anytime feasibility, a major feature that dual methods do not generically provide.

V. Implementation and algorithm tuning

In this section, we discuss two major points in the design of FD-ADMM. First, we precise and justify the choice of the procedure PROJECTION, in line 7 of FD-ADMM Algorithm 2. Next, we derive an explicit adaptive update of the reciprocal penalty parameter λ that permits to accelerate the convergence of FD-ADMM on any instance.

A. Projection procedure: A discussion

In Section IV, we advocated a link-wise separation of the formulation because it is non-trivial to project an arbitrary point onto an arbitrary closed convex polyhedron. However, the projection onto the sets S_j (see Section IV-A) can be done with an exact method with a complexity dominated by the one of sorting a list of the size of its dimension. In average, sorting a list of length q is done in $O(q \log q)$. Hence, by operating instead a link-by-link projection, the controllers save a huge amount of time by providing an (generically infeasible) approximate projection point z_{pr} and deriving a locally feasible allocation z_{*pr} (see Algorithm 2 line 10). Although the quality of the global iterate z_* may be altered by further distribution of the projection, the point is quickly generated. Paradoxically enough, FD-ADMM therefore fully adapts to any network distribution into domains *because* it functions by link, regardless of the network partition into domains. The algorithm we use for PROJECTION in FD-ADMM is presented for instance in [5] in which the authors also give a correctness proof and performance demonstration. It permits to provide an efficient update for each domain J_p.

B. Estimating the optimal parameter λ

It is well-known that the reciprocal penalty parameter λ highly conditions the convergence speed of ADMM. An inaccurate tuning can indeed lead to a very slow convergence. For appropriate problems, it is possible to use a result proven in [6] to compute an optimal reciprocal penalty parameter, that we here report. It will help us tune FD-ADMM to optimize its convergence performance[5].

Theorem 1 ([6])**.** *Assume that the following problem:*

$$\begin{aligned} &\min F(x) + G(z) \\ &\text{s.t. } Mx - Pz = 0 \end{aligned} \tag{M}$$

has a saddle point, and both objective functions are convex. Assume that M has full row rank, and that F is σ-strongly convex and has a L-Lipschitz gradient. Then, the sequence of iterates (primal and dual concatenated) of ADMM converges linearly with rate $(1+\delta)^{-1}$, where[6]

$$\delta = 2\left(\frac{||M||^2}{\lambda\sigma} + \frac{L\lambda}{\lambda_{\min}(M^{\mathrm{T}}M)}\right)^{-1}$$

and $\frac{1}{\lambda}$ is the penalty parameter in the augmented Lagrangian form (see Section III-B).

The following result directly follows.

Corollary 1. *The optimal reciprocal penalty parameter is*

$$\lambda_* = \sqrt{\frac{||M||^2\lambda_{\min}(M^{\mathrm{T}}M)}{\sigma L}}.$$

In order to be able to apply Corollary 1, we still need to express the coefficients of interest σ, L_d. Due to page limitation, we defer their calculus to [1] and we here report the result.

Fact 2. *The function $g = \bigoplus_r g_r$ is σ-strongly convex and has L_d-Lipschitz gradient with:*

$$\sigma = \alpha \min_r \frac{w_r}{B_r^{\alpha+1}}, \quad L_d = \alpha \max_r \frac{w_r}{d_r^{\alpha+1}}$$

on any compact subset of $\mathrm{dom}(g)$ of the form $K_d = \{x \geq d, Ax \leq C\}$, for $d \gg 0$, and $B_r = \min_{j\in r} C_j$.

Unfortunately, Corollary 1 cannot be directly applied to our general consensus formulation. Indeed, its matricial formulation does not provide a full-row rank matrix M. The problem which the Theorem 1 applies to is actually the original, centralized one in (3-4). Therefore, *we will derive a reciprocal penalty parameter selection for the centralized problem, and use it as a tool to estimate a satisfactory parameter for FD-ADMM.*

However, the last difficulty we encounter in choosing the optimal reciprocal penalty parameter is to correctly evaluate the Lipschitz modulus. Unfortunately, ∇g is *not* Lipschitz on the feasibility set, because of the singularity of each g_r at 0. In order to circumvent this problem, we introduce the classic concept of *disagreement point* d, according to *bargaining theory* terminology. A disagreement point d represents the minimal values for an allocation of each route. This allows to reduce the feasibility set to a compact subset of the form $K_d, d \gg 0$, on which ∇g is now Lipschitz. The disagreement point can be naturally defined as the feasible point z_* at the first iteration. Generically, there is no *a priori* guarantee that the set K_{z^*} contains the optimum, but, we remark that at least in the first iterations, the use of z_* provides a good approximation of the best reciprocal penalty parameter. The analytical evaluation of this phenomenon goes beyond the scope of this paper and we keep it for future work.

Thus, finally, we update λ in an adaptive fashion in the beginning of the algorithm with the help of those points. We found empirically that operating such update only at the initial steps of FD-ADMM and then fixing λ for the rest of the execution provides a good performance in terms of convergence speed. In the next section, we describe this

[5]We recall that a differentiable function $f : \mathbf{R}^n \to \bar{\mathbf{R}}$ is *strongly convex* with modulus σ if $(\nabla f(x) - \nabla f(y))^{\mathrm{T}}(x-y) \geq \sigma||x-y||^2, \quad \forall x, y \in \mathrm{dom}(f)$. Moreover, f is *Lipschitz* with modulus L if $|f(x)-f(y)| \leq L|x-y|, \quad \forall x, y \in \mathrm{dom}(f)$.

[6]$\lambda_{\min}$ is the smallest eigenvalue of a positive matrix, and $||M||$ is the operator norm

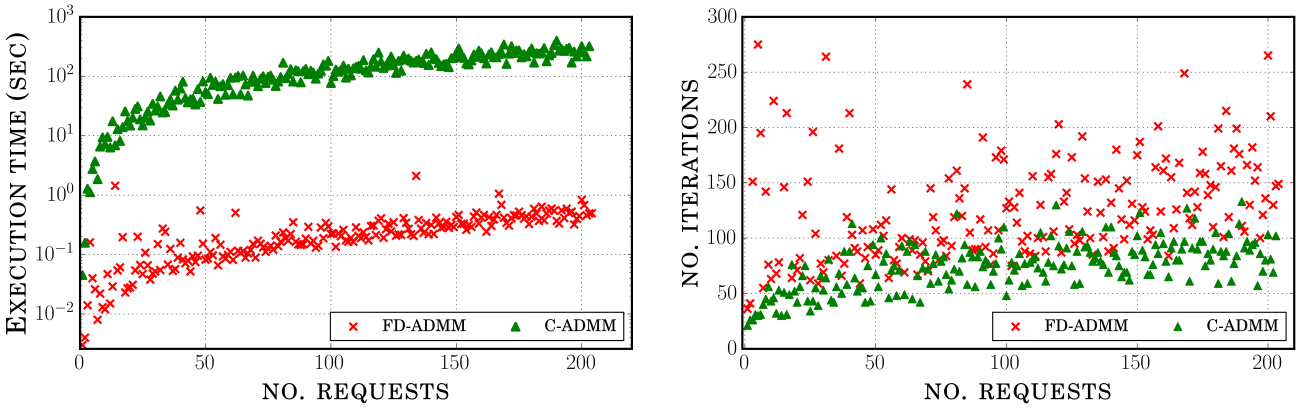

Fig. 1: C-ADMM against FD-ADMM: execution time and iteration count.

typical phenomenon in Figure 2. In all our simulations, we use the simple following update scheme to estimate the optimal penalty parameter at each execution of the algorithm.

Scheme 1 (Reciprocal Penalty Adaptation)**.** Set threshold τ. At all iterations below τ, denote by p the last output of a feasible point. Then, choose the new reciprocal penalty parameter as:

$$\lambda_* = \frac{1}{\alpha}\left(\min_{r\in R}\frac{w_r}{B_r^{\alpha+1}}\max_{r\in R}\frac{w_r}{p_r^{\alpha+1}}\right)^{-\frac{1}{2}}.$$

After τ iterations, do not update λ.

In our numerical evaluations we will set $\tau = 30$. Thus, FD-ADMM is now fully tuned and we are ready to demonstrate its performance in the next section, in terms of convergence speed in real-time scenarios.

VI. Performance analysis

We now evaluate numerically FD-ADMM in terms of its convergence properties. More specifically, in Section VI-A we compare the performance of FD-ADMM and C-ADMM in offline scenarios where the optimum is desired. In Section VI-B we evaluate FD-ADMM in real-time scenarios, where good and feasible solutions are needed on-the-fly as weights w_r vary over time. In order to benchmark the transient properties of FD-ADMM we use the standard Lagrangian dual decomposition approach (LAGR) for single-path routing in [25, 15, 19], that we recall in Algorithm 3. We here assume that domain controllers operate in synchronous mode. In this case, the decomposition into domains has no impact on FD-ADMM performance, as projection is on a link-basis. All simulations are made for the proportional fairness objective functions ($\alpha = 1$). We used the proximal operation formulas found in [3]. The algorithms under investigation were evaluated using BT's 21 CN network topology[7], containing 106 nodes and 474 links. The requests were generated by computing the shortest path between randomly chosen sources and destinations.

[7]We would like to thank the authors of [15] for their willingness to share the BT 21 CN topology dataset.

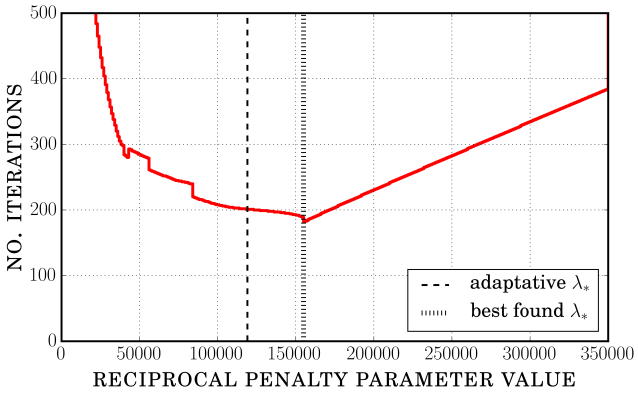

Fig. 2: Convergence rate of FD-ADMM vs. reciprocal penalty parameter. The adaptive approximation demonstrates sufficient accuracy.

A. Algorithm design

Evaluating the alleviation of the compute-intensive parts of C-ADMM was a key concern to motivate and validate the distribution to FD-ADMM. To this aim, we show in Figure 1 the computation time and iteration count for those two algorithms on small instances for a number of requests ranging from 1 to 200. The centralized projection in C-ADMM is executed using the variation of Hildreth's projection algorithm on general polyhedra in [9]. When convergence is desired, a precise stopping criterion for FD-ADMM is available, as the optimality gap can be upper-bounded by the primal and dual residuals, see [3]. In our case, evaluating those residuals results in computing the absolute variation of two consecutive values

Algorithm 3 Lagrangian-based gradient descent (LAGR)

Input: Initial positive values u_j
while a suitable termination condition is not met **do**
$x_r \leftarrow \arg\max_{x\geq 0}\{f_r(x) - x\sum_{j:j\in r} u_j)\} \quad \forall r$
$u_j = u_j - \frac{u_j}{2C_j}(C_j - \sum_{r:j\in r} x_r) \quad \forall j$
end while

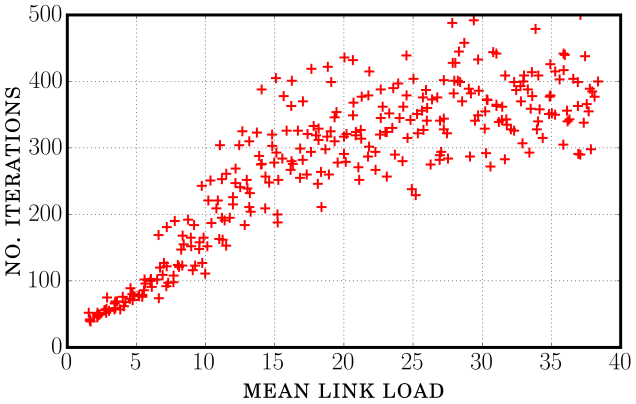

Fig. 3: Iteration count for FD-ADMM vs the mean link load (average value of the $|R_j|$).

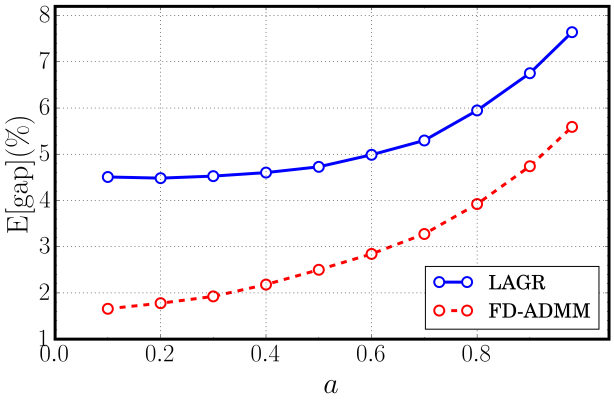

Fig. 4: Average optimality gap E[gap] vs. the variation amplitude a.

of $\tilde{z}$, and the consensus accuracy[8] $\max_{r\in R, j\in r} |z_{jr} - \tilde{z}_r|$. This is a first advantage for FD-ADMM implementation as no robust stopping criterion is available for standard gradient descent. When an optimality gap is computed, we thus consider a 10^{-6}-approximation by FD-ADMM as the reference for all tested algorithms. In Figure 2, we illustrated, on a small instance with 200 requests, the number of iterations of FD-ADMM to reach convergence for a various number of the parameter values, in order to evaluate our *adaptive* scheme's accuracy with respect to the empirically *best found* parameter. It shows that our approximation of λ_* is fairly satisfactory. In Figure 1, FD-ADMM shows that distributing the consensus over the links exchanges several more iterations for a reduction of the compute time by two orders of magnitude for small instances. Hence, the distribution does not seem to cost too much convergence rate. Not surprisingly, the use of a central projection sub-routine makes C-ADMM impossible to scale. The convergence criterion used in Figures 1 and 3 is modest (10^{-1}). Finally, we plotted a notable behavior of FD-ADMM in Figure 3. One can imagine a link between the convergence rate and the mean link load, i.e., $\frac{1}{|J|}\sum |R_j|$. This conjecture requires further investigation that we keep for future work.

B. Comparison against Lagrangian method

We now compare the proposed FD-ADMM algorithm against the classic LAGR Algorithm 3, see [25, 15, 19]. To this aim we perform two experiments, in real-time and static scenarios, respectively.

We start by evaluating the real-time responsiveness of FD-ADMM by considering a small scenario where 200 routes are established and the weights $(w_r^t)_{r\in R, t\in 0\dots T}$ vary over discrete time t, following the formula:

$$w_r^{t+1} \in [(1-a)w_r^t, (1+a)w_r^t] \quad a \in [0,1],$$

where at each event t, w_r^t is chosen uniformly within the above interval in which a determines the amplitude of the weight variation. In Figure 4 we illustrated the average optimality gap of the two algorithms achieved over 20 events with 10 iterations between each event. We observe that FD-ADMM outperforms LAGR in terms of optimality gap, although the performance of both algorithms is fairly acceptable. However, remarkably, FD-ADMM remains always feasible whereas LAGR constantly violates the constraints as weights w_r change in real-time. Figure 5 shows the percentage of constraints of the problem that are violated for each value of the amplitude a. In fact, LAGR iteratively approaches the fair resource allocation from the outside of the feasible set. This drawback is commonly amended by projecting the solution onto the feasible set. However, this is not doable in our distributed setting, as projection requires costly on-the-fly operations that require full topological information. For such reasons, we claim that the standard LAGR algorithm is not well suited for computing real-time fair allocations in a distributed SDN setting.

In our last experiment we test the two algorithms under a static scenario, where the weights w_r do not vary over time and LAGR has enough time to find at least one feasible solution. In Figure 6 we compare the optimality gap of the *best feasible* solutions found after 5 seconds runtime by FD-ADMM and LAGR, for different instance sizes over BT topology. We observe that FD-ADMM obtains a close-to-optimal feasible solution for all the instance sizes (from 100 to 6000 requests),

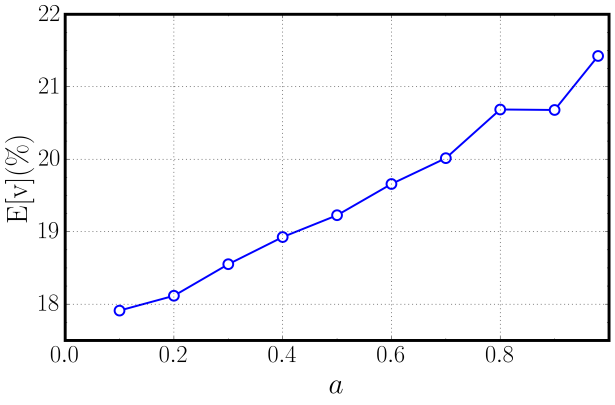

Fig. 5: Average percentage of violated constraints E[v] by LAGR vs. the variation amplitude a.

[8]One can choose any other norm in $\bigoplus \mathbf{R}^{|R_j|}$.

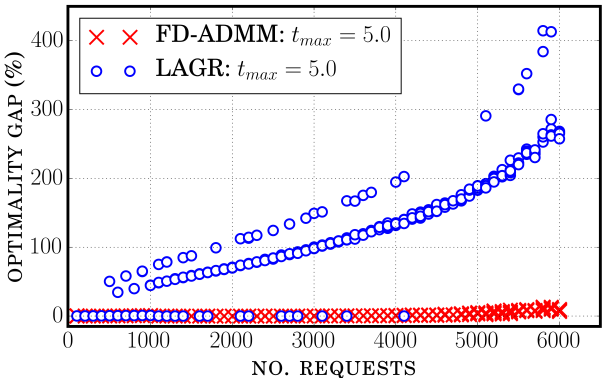

Fig. 6: Optimality gap of the best feasible point found after 5 seconds runtime.

while LAGR is still far from the optimum especially when the instance becomes large.

To recap, in this section we have demonstrated by experimentation that FD-ADMM reacts quickly to unpredictable network variations, while preserving the feasibility of the solutions computed iteratively. We then claim that FD-ADMM is a good candidate for real-time fair resource allocation in distributed SDN scenarios.

VII. Conclusions and Future Work

In this paper we addressed the real-time fair resource allocation problem in the context of a distributed SDN control plane architecture. Our main contribution is the design of a distributed algorithm that continuously generates a sequence of feasible solutions and adapts to any partitioning of the network into domains. We reformulated the α-fair resource allocation problem in the fashion of a general consensus problem to derive the FD-ADMM algorithm. This algorithm can be massively parallelized on several processors that manage different regions of the network, hence fully benefiting from the computing resources of SDN controllers in distributed architectures. We also provided a strategy for a near-optimal estimation of the penalty parameter of FD-ADMM that boosts its convergence. Finally, we compared FD-ADMM to a standard dual Lagrangian decomposition method (LAGR) and we demonstrated how the former is more adapted to a real-time situation where bandwidth has to be adjusted on-the-fly. In fact, FD-ADMM ensures a smaller optimality gap since the very first iterations and, most importantly, it produces a feasible fair allocation at all iterations.

As a next step, we envision to adapt our formulation to the case where multiple candidate paths are available for each request. Moreover, we plan to run FD-ADMM asynchronously while still guarantying near-optimal convergence rate and anytime feasibility.

Acknowledgment

This is the author edited copy of the article published in the Proceedings of ITC 29, Genoa, 4-8 September 2017.

References

[1] Allybokus, Z., Avrachenkov, K., Leguay, J., and Maggi, L. (2017). Real-time fair resource allocation in distributed software defined networks. *Inria Research Report no.9015*. available at https://hal.inria.fr/hal-01442918.

[2] Bertsekas, D. P., Gallager, R. G., and Humblet, P. (1992). *Data networks*, volume 2. Prentice-Hall International Series.

[3] Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122.

[4] Charny, A., Jain, R., and Clark, D. (1995). Congestion control with explicit rate indication. In *Proc. of IEEE ICC*.

[5] Chen, Y. and Ye, X. (2011). Projection onto a simplex. *arXiv preprint arXiv:1101.6081*.

[6] Deng, W. and Yin, W. (2016). On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing*, 66(3):889–916.

[7] Hassas Yeganeh, S. and Ganjali, Y. (2012). Kandoo: a framework for efficient and scalable offloading of control applications. In *Proc. of ACM HotSDN*.

[8] He, B. and Yuan, X. (2012). On the o(1/n) convergence rate of the douglas-rachford alternating direction method. *SIAM J. Numer. Anal.*, 50(2):700–709.

[9] Iusem, A. N. and De Pierro, A. R. (1987). A simultaneous iterative method for computing projections on polyhedra. *SIAM Journal on Control and Optimization*, 25(1):231–243.

[10] Kelly, F. P., Maulloo, A. K., and Tan, D. K. (1998). Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, 49(3):237–252.

[11] Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proc. of the IEEE*, 103(1):14–76.

[12] Lee, T.-J. and Veciana, G. D. (1998). A decentralized framework to achieve max-min fair bandwidth allocation for atm networks. In *IEEE GLOBECOM 1998*, pages 1515–1520 vol.3.

[13] Liang, C. and Yu, F. R. (2015). Distributed resource allocation in virtualized wireless cellular networks based on admm. In *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 360–365.

[14] Marasevic, J., Stein, C., and Zussman, G. A fast distributed stateless algorithm for alpha-fair packing problems. In *Proc. of ICALP), vol.55, pp.54–1, year=2016,*.

[15] McCormick, B., Kelly, F., Plante, P., Gunning, P., and Ashwood-Smith, P. (2014). Real time alpha-fairness based traffic engineering. In *Proc. of ACM HotSDN*, pages 199–200.

[16] Mo, J. and Walrand, J. (2000). Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking (ToN)*, 8(5):556–567.

[17] Mota, J. F., Xavier, J. M., Aguiar, P. M., and Püschel, M. (2012). Distributed ADMM for model predictive control and congestion control. In *Proc. of IEEE CDC*.

[18] Palle, U., Dhody, D., Singh, R., Fang, L., and Gandhi, R. (2016). PCEP Extensions for MPLS-TE LSP Automatic Bandwidth Adjustment with Stateful PCE. Internet-Draft draft-dhody-pce-stateful-pce-auto-bandwidth-09, Internet Engineering Task Force. Work in Progress.

[19] Palomar, D. P. and Chiang, M. (2006). A tutorial on decomposition methods for network utility maximization. *IEEE Journal on Selected Areas in Communications*, 24(8):1439–1451.

[20] Phemius, K., Bouet, M., and Leguay, J. Disco: Distributed multi-domain SDN controllers. In *Proc. IEEE NOMS, pages=1–4, year=2014*.

[21] Skivée, F. and Leduc, G. (2004). A distributed algorithm for weighted max-min fairness in MPLS networks. In *International Conference on Telecommunications*, pages 644–653. Springer.

[22] Stallings, W. (2013). Software-defined networks and openflow. *The internet protocol Journal*, 16(1):2–14.

[23] Sundaresan, R. et al. (2016). An iterative interior point network utility maximization algorithm. *arXiv preprint arXiv:1609.03194*.

[24] Vaughan-Nichols, S. J. (2011). Openflow: The next generation of the network? *Computer*, 44(8):13–15.

[25] Voice, T. (2006). Stability of multi-path dual congestion control algorithms. In *Proc. of Valuetools*. ACM.